NOTA 984 augustus 1977

Instituut voor Cultuurtechniek en Waterhuishouding
Wageningen

ASPECTEN VAN INFORMATIEVERWERKING

8

ENKELE METHODEN VOOR HET OP EENVOUDIGE EN OVERZICHTELIJKE WIJZE SYSTEMATISCH AANPASSEN RESP. VERANDEREN VAN FUNCTIES EN PARAMETERWAARDEN IN COMPUTER-PROGRAMMA's

dr. Ph.Th. Stol

A S P E C T E N V A N I N F O R M A T I E V E R W E R K I N G

Deel	Titel	Auteur	Nota	Datum
1	Computerverwerking van lange reeksen getallen	J.B.H.M. van Gils	935	nov. 1976
2	Optimaliseren van parameters: Het gereedmaken van een functie voor toepassing in NLV	Ph.Th. Stol	943	febr. 1977
3	Registratieverwerking voor automatische pF-bepalingen	J.B.H.M. van Gils	949	febr. 1977
4	Het systematisch bepalen van de afgeleiden van een functie ten behoeve van hun programmering	Ph.Th. Stol	948	febr. 1977
5	Het samenstellen van een input tape voor een elektrisch model	J.B.H.M. van Gils		
6	Over het samenstellen van een computerprogramma voor het optimaliseren van parameters	Ph.Th. Stol	951	apr. 1977
7	De onbekend-kode in een datafile	mevr.G.van den Berg-Buitenhuis	989	juli 1977
8	Enkele methoden voor het op eenvoudige en overzichtelijke wijze systematisch aanpassen respectievelijk veranderen van functies en parameterwaarden in computerprogramma's	Ph.Th. Stol	984	aug. 1977

De nota's handelende over Aspecten van Informatieverwerking bevatten inlichtingen over de ontwikkeling van de informatieverwerking binnen het Instituut. Naast meer concluderende en toelichtende beschouwingen zal aandacht worden besteed aan het gebruik van programma's en programmapakketten en zullen zakelijke inlichtingen over praktijkervaring met en toepassing van de informatieverwerking worden gegeven

I N H O U D

1. INLEIDING

Bij het uitvoeren van numerieke bewerkingen kan het voorkomen dat een gegeven rekenmodel of een functie achteraf een aantal malen met verschillende parameterwaarden of verschillende waarden voor de lopende variabelen moet worden doorgerekend.

Afhankelijk van de omvang van het werk, de omvang van het programma, de status van het programma en de status van het onderzoek, kan men voor een bepaalde werkwijze kiezen.

In deze nota zullen een aantal alternatieve mogelijkheden voor de oplossing van dit probleem toegelicht worden en van commentaar worden voorzien. De bedoeling hiervan is de in de praktijk nuttig gebleken mogelijkheden inclusief de job-opbouw vast te leggen voor eventuele verdere toepassing.

Tevens zal er in deze nota van worden uitgegaan dat ook voor de door te rekenen functie een aantal alternatieven aanwezig zijn zodat het programma en de job-opbouw ook in dit opzicht een zekere mate van flexabiliteit moet bezitten.

De beschreven bewerkingen werden uitgevoerd op een Control Data computer. De opbouw van de benodigde jobs voor de besproken werkwijzen zullen worden toegelicht en met volledige, geteste, voorbeelden worden geïllustreerd.

2. TERMINOLOGIE

In het volgende zal de te veranderen factor aangeduid worden met 'parameter'. Aangenomen wordt dan dat een 'functie' - in de meest algemene zin van het woord - met verschillende waarden van

een of meer parameters moet worden getest. Verondersteld kan nog worden dat het gehele rekenproces reeds geprogrammeerd is, waarbij het programma aangeduid zal worden met 'hoofdprogramma'.

In het hoofdprogramma moet de mogelijkheid bestaan 'alternatieve functies' of functie-onderdelen in te bouwen en deze met 'alternatieve parameterwaarden' door te rekenen.

Aangezien voor het aangeven van de wijze 'waarop' de parameters en functies zelf er niet toe doen, zullen de volgende verzonnen elementen als voorbeeld dienen. Te gebruiken namen in FORTRAN (en wiskunde) zijn:

Te variëren parameters: P, P1, P2, (p, p_1, p_2);
Geïndicieerd: P(I), (p_i);

Te berekenen functiewaarde: Y, (y);
Functie-onderdeel: XVALUE, (x);

Het hoofdprogramma wordt genoemd: MAIN;

Te variëren parameters heten samen: DOTHIS;

Te variëren functie-onderdeel:
. omschrijving is opgenomen in TEXT;
. theoretisch gedeelte is opgenomen in: THEORY.

Met de term 'UPDATE' wordt het CDC-systeem aangeduid waarmee informatie op files kan worden bijgewerkt, verbeterd, respectievelijk kan worden veranderd. Een omschrijving van UPDATE is gegeven in Nota 951: Aspecten van Informatieverwerking No 6: Over het samenstellen van een computerprogramma voor het optimaliseren van parameters (STOL, 1977).

3. RANGSCHIKKING VAN MOGELIJKHEDEN

Oplossingen tot het gestelde probleem zullen in een enigszins systematische volgorde besproken worden. Hierbij is de nadruk komen te liggen op de mate van gecompliceerdheid waarmee in het oorspronke-

lijke computerprogramma moet worden inbegrepen om het gewenste resultaat te bereiken.

Er wordt namelijk van de gedachte uitgegaan dat de wens tot doorrekenen met alternatieve waarden vaak ook zal ontstaan bij het gebruik van gereed zijnde programma's waarbij men na de eerste resultaten te hebben bestudeerd, meer inzicht in de numerieke eigenschappen van de functie wil verwerven.

De toegepaste volgorde van behandeling van enkele mogelijkheden is globaal van g e c o m p l i c e e r d naar e e n v o u d i g met betrekking tot wijzigingen die in het bestaande hoofdprogramma moeten worden aangebracht. Het voorgaande kan wél betekenen dat 'eenvoudig' inhoudt dat de job-opbouw meer gecompliceerd wordt. Wanneer echter eenmaal uitgekiend is hoe de jobs in elkaar gezet moeten worden en op elkaar moeten aansluiten, kan het ontworpen systeem gemakkelijk ook op andere gevallen worden toegepast. De hoeveelheid werk is dan aanmerkelijk minder dan wanneer het hoofdprogramma zelf steeds moet worden aangepast. Het vastleggen van een goed werkend systeem is de bedoeling van deze nota.

4. INVLOED VAN HET ONTWERP

Een aantal, reeds in de inleiding besproken, factoren bepalen de richting waarin naar een oplossing kan worden gezocht,

Is het doorrekenen van de functie met verschillende parameterwaarden een zeer omvangrijk karwei, dan zal men er de voorkeur aangeven hiervoor een zelfstandig programma of programma-onderdeel te ontwerpen.

Is het hoofdprogramma zeer klein dan is de eenvoudigste oplossing waarschijnlijk om dit steeds in de Fortran-tekst aan de nieuwe parameterwaarden aan te passen. Gezien de snelheid waarmede huidige computers een programma vertalen - en de kosten dan bij kleine programma's gering zijn -, zal het minder efficient zijn tijd te besteden aan het ontwerpen van, repectievelijk zoeken naar, een geautomatiseerde werkwijze.

Wanneer het hoofdprogramma nog in de ontwikkelingsfase is zal

men eerder een systeem toepassen waarbij van UPDATE-faciliteiten gebruik gemaakt wordt dan wanneer het hoofdprogramma reeds definitief vaststaat.

Hetzelfde geldt ook voor de status van het onderzoek. Wanneer in de opzet daarvan weinig verandering meer komt zal men eerder geneigd zijn een definitieve oplossing in het programma in te bouwen dan wanneer alles nog in ontwikkeling is en nog naar de juiste wiskundige formulering gezocht wordt.

De praktijk leert dat men vaak, mogelijk na eerst met succes het hoofdprogramma te hebben toegepast, op combinaties van parameterwaarden stuit die nader onderzoek van de gehanteerde functies vragen. De vraag zal dan zijn het hoofdprogramma tijdelijk aan een eenvoudig systeem voor het doorrekenen met alternatieve parameterwaarden aan te passen.

Ongeveer dezelfde argumenten gelden voor het doorrekenen van alternatieve functies. Wel moet worden opgemerkt dat ingrepen in een programma om dit te realiseren veel ingrijpender zullen zijn zodat reeds vanaf de eerste opzet rekening gehouden moet worden met een wijze van programmeren waarbij alternatieve functies in duidelijk te onderscheiden programma-onderdelen worden ondergebracht. Een dergelijk 'blok' kan dan in zijn geheel worden vervangen.

Bovenstaande overwegingen zijn slechts zeer globaal. In elk geval afzonderlijk zal men, door de praktijk geleid, tot een keuze moeten komen. In het nu volgende zullen hiervoor enige richtlijnen worden gegeven die van uitgewerkte voorbeelden zullen worden voorzien.

5. INDELING VAN MOGELIJKHEDEN, ALGEMEEN

Voor het gemakkelijk vervangbaar maken van programma-onderdelen kunnen een aantal algemene richtingen worden genoemd die hierna in het kort ten aanzien van hun voordelen (+) en nadelen (-) zullen worden toegelicht.

a. Het inlezen van alternatieve parameterwaarden via Fortran-READ--instructies:

+) Toepasbaar voor elke waarde van de parameter
+) Kaartendek hoofdprogramma (dat omvangrijk kan zijn) en kaartendek met parameterwaarden kunnen afzonderlijk worden behandeld
+) Geringe file administratie

-) Gebonden aan vast FORMAT
-) Geen indicatie op de in te lezen kaarten omtrent de parameter zelf
-) Niet toepasbaar voor het veranderen van functies of functie--onderdelen

b. Het veranderen van de programmatekst in het kaartendek van het hoofdprogramma:

+) Toepasbaar voor elke waarde van de parameter
+) Toepasbaar voor het inzetten van alternatieve functies
+) Op de kaarten voor nieuwe parameterwaarden komt de parameter zelf voor (b.v. P = 0.5) hetgeen het in combinatie wisselen van meer parameters overzichtelijk maakt.

-) Er moet met het gehele kaartendek gemanipuleerd worden
-) Het programma moet steeds opnieuw ingelezen en vertaald worden
-) Er is geen administratie van aangebrachte wijzigingen in de formules tenzij de gehele programmatekst van het hoofdprogramma tijdens het vertalen opnieuw wordt geprint

Opgemerkt wordt nog dat dit laatste bezwaar niet geldt voor de parameterwaarden aangezien deze in een geschikt FORMAT automatisch in de output zichtbaar gemaakt kunnen worden, zodat altijd is na te gaan welke combinatie van parameterwaarden werd toegepast.

c. Het veranderen van de programmatekst door het UPDATE-en van het hoofdprogramma:

+) Toepasbaar voor elke waarde van de parameter
+) Toepasbaar voor het inzetten van alternatieve functies
+) Administratie van aangebrachte wijzigingen
+) Geen manipulatie met gehele kaartendek

-) Herhaald inlezen en vertalen van het gehele programma
-) File-administratie

d. Het veranderen van de programmatekst in het kaartendek van een subroutine:

+) Toepasbaar voor elke waarde van de parameter
+) Toepasbaar voor functies
+) Geen manipulatie met gehele kaartendek

-) Geen administratie van aangebrachte wijzigingen van formules tenzij de programmatekst van de subroutine wordt geprint (wat overigens slechts een werk van zeer beperkte omvang zal zijn)

Inbegrepen in het UPDATE-systeem is steeds de mogelijkheid om van a l l e aangebrachte wijzigingen een afschrift in de output te verkrijgen. Hiervoor moet een geschikte waarde van de list-parameter L in de UPDATE-instructiekaart worden gekozen. Teneinde de uitvoer te beperken zal men veelal de optie L = 0 gebruiken. In het navolgende zal echter behandeld worden hoe men dan toch een zekere administratie omtrent het type van de toegepaste formules kan verkrijgen door een juiste toepassing van *COMDECK-kaarten (hoofdstuk 8.2.1).

6. INDELING VAN MOGELIJKHEDEN, SPECIFIEK

Reeds is aangegeven dat de mogelijkheden gerangschikt zullen worden in de volgorde waarbij steeds minder in het hoofdprogramma zelf behoeft te worden ingegrepen. Achtereenvolgens zijn dit:

A. Gebruik van besturingsparameters voor het definiëren van alternatieve mogelijkheden
B. Gebruik van DO-loops gecombineerd met geïndiceerde parameters en het inlezen van parameterwaarden
C. Gebruik van DO-loops voor het genereren van parameterwaarden
D. Gebruik van herhaald UPDATE- en van het hoofdprogramma en het vertalen daarvan
E. Gebruik van een subroutine voor alternatieve parameterwaarden zonder herhaald vertalen van het hoofdprogramma

Het zal in het volgende duidelijk worden dat verschillende mogelijkheden kunnen worden gecombineerd. Hierop zal niet in detail worden ingegaan.

7. BESTURINGSPARAMETERS

Als eerste mogelijkheid is in hoofdstuk 6 het gebruik van besturingsparameters genoemd. Hiervan zullen alleen enkele voor- en nadelen worden genoemd.

+) Alle gewenste veranderingen kunnen met één codegetal worden opgegeven
+) Het opgeven van veranderingen vraagt slechts een minimaal aantal handelingen, namelijk de invoer van de code

-) De gehele procedure van alternatieve functies en hun wisselend gebruik moet vast worden ingebouwd en van tevoren volledig worden geprogrammeerd
-) Alle combinaties van toe te passen parameterwaarden moeten worden geprogrammeerd en van een code worden voorzien
-) Er moet enige administratie worden bijgehouden omtrent de betekenis van de codes

Voor een voorbeeld wordt verwezen naar nota 943 (STOL, 1977a).

8. DO-LOOPS

Door middel van DO-loops wordt automatisch een programma-onderdeel een aantal nader te bepalen maken doorlopen. Door in principe een geheel programma te 'omranden' met een DO-loop wordt het programma in zijn geheel opnieuw herhaald.

Als voordelen kunnen worden genoemd:

+) Het alternatief doorrekenen kan vást worden ingebouwd en herhaaldelijk worden gebruikt
+) Het aantal malen dat herhaald doorrekenen moet worden toegepast kan in het programma worden gedefinieerd, dan wel apart worden ingelezen
+) Het geven van de juiste alternatieve waarden aan de parameters kan soms met behulp van de DO-lus tot stand worden gebracht

En als nadelen gelden:

-) In een bestaand hoofdprogramma zal het 'omranden' met DO-loops zorgvuldig moeten plaatsvinden. Met name moet een oplossing gezocht worden voor de situatie wanneer een rekenproces op verschillende manieren kan worden beëindigd. Voor het continueren van de DO-loop zal men bij al deze uitgangsmogelijkheden een sprong naar het eind van de betreffende DO-loop moeten inbouwen
-) Voor elke te variëren parameter zal een afzonderlijke DO-loop moeten worden ingebouwd, de eindpunten van de verschillende loops behoeven niet noodzakelijkerwijs dezelfde te zijn, wat het geheel weer extra gecompliceerd maakt
-) Indien het alternatief doorrekenen met verschillende parameterwaarden slechts incidenteel behoeft plaats te vinden zal de ontworpen mogelijkheid weer uitgeschakeld moeten kunnen worden. Dit kan overigens door begin- en eindpunt van de DO-loop gelijk te kiezen. Het betekent wel dat ingrijpen 'van buitenaf' in de definitie van de loop noodzakelijk blijft

-) Worden alternatieve parameterwaarden ingelezen en geïndiceerd weggezet, dan zal de nodige administratie en organisatie moeten worden verzorgd om het gehele systeem flexibel te houden naar het aantal parameters dat moet worden gevarieerd en het aantal waarden dat per parameter moet worden toegepast
-) Indien parameterwaarden in het programma zelf worden gedefinieerd moet het programma steeds opnieuw vertaald worden. Hoewel wat tijd en kosten betreft dit geen groot nadeel behoeft te zijn zal het manipuleren met een groot pak kaarten voor het veelvuldig aanbrengen van kleine veranderingen als een bezwaar worden gevoeld

8.1. Voorbeeld 1: D O - l o o p m e t g e ï n d i c e e r d e p a r a m e t e r

De DO-loop van een eerste waarde (i-first: IFST) tot een laatste waarde (i-last: ILST) welke waarden in het programma moeten worden gedefinieerd respectievelijk moeten worden ingelezen.

Uit dit voorbeeld komen voor het flexibel toepassen van herhaald doorrekenen de volgende nadelen naar voren
-) De invoer van een nieuwe reeks parameterwaarden is aan een strak Format gebonden. Men zal steeds moeten verifiëren of aan dit Format is voldaan
-) Op de input-kaarten valt niet af te lezen welke parameter is gedefinieerd: de inputkaarten vermelden slechts getallen

```
Regel--1234567890 = kaartkolom

 1          .
 2          .
 3          .
 4    C
 5          DIMENSION P(5)
 6          READ(2,20) (P(I),I=1,5)
 7       20 FORMAT(5F5.0)
 8          IFST=1  $  ILST=5
 9          .
10          .
11          .
12          DO 10 I=IFST,ILST
13          .
14          .
15          .
16          Y=P(I)*XVALUE
17          .
18          .
19          .
20       10 CONTINUE
21    C
22          .
23          .
24          .
-------1234567890 = kaartkolom
```

Voorbeeld 1. Enkele statements uit een Fortran-programma waarin 5 alternatieve parameter waarden p_i worden ingelezen en in DO-loop 10, achtereenvolgens worden toegepast (zie toelichting hiernaast)

Regel	Toelichting bij Voorbeeld 1
1,2	enz. niet nader omschreven Fortran-opdrachten
4,21	Comment-kaarten, in dit geval een blanke regel spatie
5	Er worden voor de parameter P in totaal 5 posities in het geheugen gereserveerd
6	Volgens kaartindeling omschreven in Format 20 werden achtereenvolgens 5 waarden voor P ingelezen. De kaartlezer is aangeduid met de code 2
7	De 5 waarden voor P beslaan maximaal 5 kolommen waarin de waarde van P, rechts aangeschoven, moet worden geponst. De getallen worden opgevat te hebben nul cijfers achter de komma
8	Begin (i-first) en eindpunt (i-last) van de DO-loop worden gedefinieerd
12	Beginpunt van de DO-loop
16	Veronderstelde noodzakelijke berekening met alternatieve parameterwaarden
20	Eindpunt van de DO-loop

Opmerking: Begin en eindpunt van de DO-loop kunnen zonodig ook ingelezen worden op een aparte kaart of op dezelfde kaart als de parameterwaarden.

In de READ-opdracht is aangenomen dat de INPUT-file in de programma-kaart gedefinieerd is als TAPE 2

8.2. Voorbeeld 2: D O - l o o p m e t g e g e n e r e e r d e p a r a m e t e r w a a r d e n

Wanneer de achtereenvolgens te gebruiken parameters waarden doorlopen die in een bepaalde relatie tot elkaar staan, kan nagegaan worden of het mogelijk is deze waarden met behulp van een DO-loop te genereren.

In het nu volgend voorbeeld wordt eenvoudigheidshalve aangenomen dat de parameterwaarden uit een lineaire combinatie van de DO-loop index I kan worden verkregen. In het voorbeeld neemt I de waarden aan: 2, 4, 6, 8 en 10. In de uitdrukking voor P wordt de subroutine FLOAT gebruikt teneinde de integer I in één bewerking met de continue variabelen A en B op te kunnen nemen. Uiteraard moeten waarden voor A en B van tevoren in het programma zijn gedefinieerd, respectievelijk worden ingelezen.

De voordelen van deze werkwijze zijn:

+) Het inlezen van de parameterwaarden vervalt en er behoeft hiervoor geen administratie bijgehouden te worden

+) Op eenvoudige wijze, zonder aanpassing van bijvoorbeeld de DIMENSION, kan de reeks parameterwaarden ongelimiteerd worden uitgebreid door het eindpunt van de DO-loop 'verder' te kiezen ofwel de stapgrootte te verkleinen

Enkele nadelen zijn:

-) Het systeem is weinig flexibel

-) Begin- en eindpunt en stapgrootte van de DO-loop moeten steeds worden ingelezen, in ieder geval worden gedefinieerd

-) De methode is alleen toepasbaar voor parameterwaarden die uit een niet al te gecompliceerde transformatie van I kunnen worden verkregen

-) Indien met de stapgrootte wil wijzigen na een vooraf bepaald aantal berekeningen, zal extra programmeerwerk noodzakelijk zijn en het systeem weer ingewikkelder worden

Het laatste geval doet zich voor bij progressieve reeksen, waarvan een bekend voorbeeld het aantal dagen is waarover de neerslag moet worden gecumuleerd. Wordt de stapgrootte tussen haakjes vermeld dan kan men denken aan de reeks (in dagen)

k = 1(1)5(5)30(30)360(180)1800

of iets dergelijks.

```
Regel--1234567890 = kaartkolom

 1          .
 2          .
 3          .
 4    C
 5          IFST=2  $  ISTEP=2  $  ILST=10
 6          .
 7          .
 8          .
 9          DO 20 I=IFST,ILST,ISTEP
10          .
11          .
12          .
13          P=A*FLOAT(I)+B
14          Y=P*XVALUE
15          .
16          .
17          .
18       20 CONTINUE
19    C
20          .
21          .
22          .

-------1234567890 = kaartkolom
```

Voorbeeld 2. Enkele statements uit een Fortran-programma waarin voor i=2, 4, 6, 8, 10 in DO-loop nr 20, alternatieve waarden voor de parameter P worden toegepast (zie toelichting hiernaast)

Regel	Toelichting bij Voorbeeld 2
1,2	enz. niet nader omschreven Fortran-opdrachten
5	Begin (i-first), stapgrootte (i-step) en eindpunt (i-last) van de DO-loop worden gedefinieerd
9	Beginpunt van de DO-loop
13	Lineaire transformatie van de DO-index I voor het verkrijgen van de alternatieve waarden voor de parameter P. De constanten A en B moeten vooraf gedefinieerd zijn. De integer I wordt met de hulproutine FLOAT omgezet in een continue variabele (floating point variable)
14	Veronderstelde noodzakelijke berekening met alternatieve parameterwaarde P, welke in 13 gedefinieerd is
18	Eindpunt van de DO-loop

Opmerking: Begin, eindpunt en stapgrootte van de DO-loop kunnen zonodig ook ingelezen worden evenals de waarden voor de transformatieconstanten A en B.
De transformatie-opdracht (13) zou vervangen kunnen worden door een andere formule Hiervoor kan een oplossing met UPDATE (zie COMDECK DOTHIS in de volgende voorbeelden) of met een subroutine (zie SUBROUTINE DOTHIS in de volgende voorbeelden) gekozen worden.

9. UPDATE

Met het UPDATE-systeem wordt de administratie van aan te brengen veranderingen in een programma automatisch door de computer verzorgd. Een korte omschrijving van UPDATE op praktijkbasis wordt gegeven door STOL (1977).

In het kort komt het erop neer dat met een eerste zogenaamde creation-run informatie (data, een programma, een programma-onderdeel) op een NPL-file (new program library) wordt geplaatst. Met een zogenaamde correction-run wordt de informatie uit het systeem weer opgehaald, de file heet dan OPL-file (old program library) en heet na te zijn verbeterd, uitgebreid, aangepast, veranderd, weer NPL-file. De namen NPL en OPL zijn algemene aanduidingen, in de uitvoering van desbetreffende computerjobs kunnen namen voor deze files verzonnen worden bijvoorbeeld voor de NPL: N = NEW! of voor de OPL; P = OLD!.

Voor herhaald gebruik, in verschillende achtereenvolgens uit te voeren jobs, moeten de files beschikbaar blijven en dus permanent worden gemaakt met een REQUEST- en een CATALOG-instructie.

Alvorens te kunnen worden gebruikt moeten de permanent files weer local files worden hetgeen bewerkstelligd wordt met een ATTACH--instructie. In de voorbeelden wordt dit verder toegelicht.

De voordelen, verbonden aan het UPDATE-en zijn:

+) De alternatieve parameterwaarden kunnen als Fortran tekst worden opgenomen en zijn dus niet aan een strak Format gebonden

+) Door het hoofdprogramma op een UPDATE-file te plaatsen en als permanent file op te slaan behoeft slechts met een klein pakket kaarten te worden gemanipuleerd, namelijk alleen de kaarten met parameterwaarden en een aantal jobbesturingskaarten

+) Door in Fortran tekst te werken is het mogelijk alle in combinatie te kiezen parameterwaarden in één dek bijeen te houden en de volgorde van kaarten binnen dit dek te permuteren; alleen de laatste waarde gegeven aan de parameter zal worden gebruikt

Als nadelen kunnen worden genoemd:

-) Het kaartendekje zal na elke permutatie van parameterkaarten opnieuw moeten worden ingelezen. Zonder 'zelfbediening' in het computercentrum kan jobverwerking dan een tijdrovende bezigheid zijn

-) Er moet enige file-administratie worden verzorgd, doch het nadeel hiervan beperkt zich tot het opzetten van het systeem. Worden de name aan de files zo gegeven dat een roulerend systeem, zoals in de voorbeelden weergegeven, wordt opgebouwd, dan is dit nadeel volledig overwonnen

N.B. Met een roulerend systeem van file-administratie wordt bedoeld dat niet steeds opnieuw namen aan files moeten worden gegeven, dat files moeten worden verwijderd of aangemaakt enz. Kortom: de file-administratie verloopt automatisch en men behoeft het dek met stuurkaarten nadat het eenmaal voor een bepaalde werkwijze is samengesteld niet meer te wijzigen. Op deze wijze zijn ook de jobs in de volgende voorbeelden ontstaan.

9.1. Voorbeeld 3: U P D A T E - e n e n h e r h a a l d v e r t a l e n v a n h e t h o o f d p r o g r a m m a

Verondersteld wordt dat de te variëren parameterwaarden zijn opgenomen in een door een programma oproepbaar UPDATE-deck (zie JOB1):

```
*COMDECK DOTHIS
      P1=300.
      P1=500.
      P2= .50
      P2= .55
      P2= .60
```

Het manipuleren met een dergelijk dek is eenvoudig. Aangezien steeds de laatste definitie geldt, is het niet nodig de overige kaarten apart te bewaren, doch kan het dek compleet blijven. Een cyclische verwisseling van kaarten die P1 definiëren, gecombineerd met die welke P2 definiëren, geven het vereiste resultaat. In boven-

staand voorbeeld wordt de functie dus doorgerekend met de parameter-combinatie (P1; P2) = (500.; .60).

Verder wordt verondersteld dat enige begeleidende en toelichtende tekst op het, denkbeeldige, onderwerp gegeven wordt in het volgende dek, bijvoorbeeld:

```
*COMDECK TEXT
C
C   EERSTE METHODE
C
C   BEREKENING MET FORMULE 1
C
      WRITE(3,15)
   15 FORMAT(1H1"FORMULE 1"//)
```

waarmee op de output (code 3 in de WRITE-statement)

```
FORMULE 1
```

wordt geprint, beginnend op een nieuwe pagina (carriage control 1H1)

De toe te passen formules, of formule-onderdelen die 'uitwisselbaar' moeten zijn, kunnen opgenomen worden gedacht in het dek

```
*COMDECK THEORY
      -
      -
      - fortran-tekst
      -
      -
```

Het zal duidelijk zijn dat de comdecks geen complete te vertalen programma's behoeven te bevatten zoals met subroutines het geval is. Wel moeten de onderdelen zo in het hoofdprogramma passen dat aan de gebruikelijke regels voor een Fortran-tekst voldaan is. Zo mag, in bovenstaande voorbeelden, het hoofdprogramma niet reeds een label no 15 bevatten (wordt in comdeck TEXT gebruikt), maar in een alternatief geval mag weer wel in bijvoorbeeld *COMDECK NEWTEXT dit label

voorkomen, dus bijvoorbeeld

```
15 FORMAT(1H1"FORMULE 2"//)
```

De volgende voorbeelden bevatten de complete computer-jobs voor het illustreren van de voorstellen. Een korte toelichtende beschrijving wordt gegeven bij elke job. Tevens wordt verwezen naar de bijbehorende figuren waarin de samenhang tussen de verschillende files nog nader wordt geïllustreerd. Het gebruik van enkele termen zoals omschreven in nota 951 (STOL, 1977) wordt bekend verondersteld.

Het voorbeeld bestaat uit 3 computer-jobs.

N.B. In de nu volgende voorbeelden wordt een asterisk (sterretje) met het teken * aangeduid.
EOR wordt geponst met de dubbel ponsing 7/8/9 in kolom 1 (End of Record)
EOF wordt geponst met de dubbelponsing 6/7/8/9 in kolom 1 (End of File).

```
Regel--1234567890 = kaartkolom

  1    JOB1,CL50000,T10,IO10.
  2    ACCOUNT,...
  3    ATTACH,MAIN,MAIN,ID=...
  4    PURGE,MAIN.
  5    REQUEST,NEW1,*PF.
  6    UPDATE,N=NEW1,C=0,W,L=0.
  7    ITEMIZE,NEW1,N.
  8    CATALOG,NEW1,MAIN,ID=...
  9    EOR (stuurkaarten-jobcontrol)
 10    *DECK MAIN
 11          PROGRAM MAIN(INPUT,...)
 12          fortran tekst hoofdprogramma
 13    *CALL DOTHIS
 14          vervolg fortran tekst hoofdprogramma
 15    *CALL TEXT
 16          vervolg fortran tekst hoofdprogramma
 17    *CALL THEORY
 18          vervolg fortran tekst hoofdprogramma
 19          END
 20    EOR (eerste update, creation run)
 21    EOF (job)

-------1234567890 = kaartkolom
```

Voorbeeld 3.1. Volledige computerjob voor het creëren van een Update-NPL van het hoofdprogramma MAIN, ondergebracht in het Update-deck MAIN. In de Fortran tekst van het programma komen oproepen van drie nader te definiëren Update-comdecks voor. Deze comdecks bevatten de Fortran teksten die het hoofdprogramma moeten gaan completeren (zie toelichting hiernaast).

Regel Toelichting bij Voorbeeld 3.1.

(zie ook fig. 1, pag. 33)

1 Jobkaart waarin de naam van de job (hier JOB1) vrij mag worden gekozen en de geheugenruimtereservering (CL), rekentijd reservering (T) en in/output-tijd reservering (IO) aan de omvang van het onderwerp moet worden aangepast. De hier gegeven waarden zullen praktisch steeds voldoen

2 Accountkaart, voor gebruik hiervan is overleg met de Afd. Wiskunde noodzakelijk

3 De bestaande file MAIN waarop het hoofdprogramma geplaatst is, wordt local gemaakt. Voor de identifier specificatie (ID) is overleg met de Afd. Wiskunde noodzakelijk

4 De combinatie van locale file MAIN met de permanent file MAIN wordt vrijgegeven voor nieuw gebruik van dezelfde (eerste) cycle van de permanent file. De file zelf gaat zolang deze job duurt nog niet verloren.

N.B. Wanneer er geen file MAIN (meer) bestaat, moeten de kaarten 3 en 4 uit het dek worden verwijderd

5 Aan het systeem wordt ruimte opgevraagd om de local file NEW1 in de permanent file opslag te kunnen opbergen

6 Het UPDATE-systeem wordt opgeroepen. Er wordt een NPL gecreëerd met file naam NEW1. De te creëren Update-file wordt gelezen uit de Input-file van EOR (end of record van de job) tot EOR (end of record van dek MAIN), en bestaat dus uit het dek MAIN. Er wordt geen programma naar de compile file geschreven (C = 0), er vindt geen output van de update-run plaats (L = 0). Wenst men deze output dan kan men de parameter L weglaten wat resulteert in alleen de belangrijkste administratieve gegevens, óf men kan L = F kiezen en een volledige output (full) inclusief de tekst van het dek MAIN verkrijgen. W is een parameter voor de file organisatie

7 Met deze instructie wordt op de output weergegeven wat de inhoud is van file NEW1. Er wordt vermeld dat de file een update-status heeft en er komt de opgave

```
DECK LIST.
          MAIN
```

wat aangeeft dat de file NEW1 inderdaad het dek MAIN bevat. De parameter N geeft aan dat alle records van de file NEW1 op deze wijze moeten worden doorgewerkt

8 De file NEW1, een local file, wordt thans opgeslagen en bewaard als permanent file en krijgt de naam MAIN

9 End of Record: sluitkaart van de jobbesturing (7/8/9-dubbelponsing in kolom 1)

10 t/m 19 Het dek MAIN dat het hoofdprogramma bevat. De Fortran tekst is nog niet compleet. In een volgende job moeten de onderdelen DOTHIS, TEXT en THEORY nader worden gedefinieerd

20 End of Record: sluitkaart van het dek MAIN

21 End of file: sluitkaart van de complete job (6/7/8/9 dubbelponsing in kolom 1).

9.1.1. Opmerkingen bij voorbeeld 3.1

In het voorgaande voorbeeld is tamelijk 'zuinig' met de file namen omgesprongen. Dit om het geheel overzichtelijk en roulerend te maken. Het is overigens niet noodzakelijk steeds dezelfde file naam te gebruiken al zijn de voordelen ervan groot. In principe zou ook correct zijn

```
ATTACH,OLD1,THATONE,ID = ...
PURGE,OLD1.
REQUEST,NEW1,*PF.
UPDATE,N=NEW1,C=0,W,L=0.
CATALOG,NEW1,THISONE,ID=...
EOR
*DECK MYOWN
      PROGRAM MAIN
.
.
etc.
```

Mogelijk is het geheel nu overzichtelijker in zijn onderscheiden eenheden, het systeem is echter niet meer roulerend daar een volgende job zal moeten beginnen met

```
ATTACH,OLD1,THISONE,ID=...
```

zodat een nieuwe kaart moet worden gemaakt.

Zou gekozen zijn voor de schrijfoptie L=F dan wordt de volledige tekst van de input geprint met de update-nummering en wel als volgt:

```
*DECK MAIN                          MAIN    1
      PROGRAM MAIN(INPUT...)        MAIN    2
      ---                           MAIN    3
      .
      .
      ---                           MAIN   24
*CALL DOTHIS                        MAIN   25
      ---                           MAIN   26
      .
      .
      .
      etc.
```

In de derde job in dit voorbeeld wordt de opdracht MAIN.25 vervangen door de Fortran tekst welke in DOTHIS is opgenomen (zie aldaar). Dit completeert dan de Fortran tekst van het hoofdprogramma.

Voor Voorbeeld 3.2. zie volgende pagina

```
Regel--1234567890 = kaartkolom

  1    JOB2,CL50000,T10,IO10.
  2    ACCOUNT,...
  3    ATTACH,CASE,CASE,ID=...
  4    PURGE,CASE
  5    RETURN,CASE
  6    REQUEST,CASE,*PF.
  7    ATTACH,MAIN,MAIN,ID=...
  8    UPDATE,N=CDS,C=0,L=F,W.
  9    UPDATE,P=MAIN,M=CDS,N=CASE,C=0,W,
 10    ITEMIZE,CASE,N.
 11    CATALOG,CASE,CASE,ID=...,MR=1.
 12    EOR(stuurkaarten-jobcontrol)
 13    *COMDECK TEXT
 14          fortran tekst
 15    *COMDECK THEORY
 16          fortran tekst
 17    EOR (eerste update, creation run)
 18    EOR (tweede update, merge run)
 19    EOF (job)

-------1234567890 = kaartkolom
```

Voorbeeld 3.2. Volledige computerjob voor het creëren van een Update-NPL van de door het hoofdprogramma MAIN op te roepen common-decks TEXT en THEORY. De comdecks bevatten de Fortran-teksten die het hoofdprogramma moeten gaan completeren en worden hiermede op één NPL (N=CASE) gezet (zie toelichting hiernaast).

Regel Toelichting bij Voorbeeld 3.2
(zie ook fig. 1, pag. 33)

1 t/m 4 Overeenkomstig als in JOB1

5 Aangezien de naam CASE weer opnieuw als local file zal gaan worden gebruikt, moet deze naam aan het systeem voor vrij gebruik worden 'teruggegeven', de inhoud van de file gaat op dit moment meteen verloren

6 Overeenkomstig als in JOB1

7 De in JOB1 aangemaakte permanent file wordt opgevraagd en local (en dus bruikbaar) gemaakt

8 Het UPDATE-systeem wordt opgeroepen. Er wordt een NPL-gecreëerd met filenaam CDS (verzonnen afkorting voor ComDeckS). De te creëren update-file wordt gelezen uit de Input-file van EOR tot EOR en bestaat dus uit twee comdecks. Er wordt geen programma naar de compile file geschreven (C=0), er wordt een volledige administratie op de output geprint (L=F, nl. Full)

9 De in 8 gecreëerde NPL-file CDS heeft nu de status van OPL-file (P=CDS). Deze file wordt samen met de update-file MAIN (gecreëerd in de eerste job) gemengd tot één update-file (merge met: M=MAIN) en de nieuwe update-file (N) wordt nu CASE genoemd. Er wordt nog niets naar de compile-file geschreven aangezien nog steeds het gedeelte DOTHIS moet worden ingevuld en de Fortran tekst dus nog niet compleet is.

Verdere stuurkaarten spreken voor zich.
De ITEMIZE-instructie heeft nu tot resultaat dat op de output verschijnt:

```
ITEMIZE OF CASE
   DECK LIST.
                MAIN        TEXT        THEORY
```

waaruit blijkt dat inderdaad alle onderdelen op de file CASE staan vermeld.

11 De file CASE wordt op permanent file geplaatst. De toevoeging MR=1 betekent dat in een later stadium verschillende jobs van dezelfde permanent file gebruik zullen gaan maken (multi-read optie). Door toevoeging van deze optie behoeven de jobs niet op elkaar te wachten en wordt een snellere verwerkingstijd gegarandeerd.

Voor Voorbeeld 3.3 zie volgende pagina

```
Regel--1234567890 = kaartkolom

  1    JOB3,CL70000,T20,IO10.
  2    ACCOUNT,...
  3    ATTACH,CASE,CASE,ID=...
  4    UPDATE,N=DOWHAT,C=0,L=0,W.
  5    UPDATE,P=CASE,M=DOWHAT,N=NEWJOB,C=0,W,L=0,
  6    UPDATE,P=NEWJOB,C=DOJOB,F,L=0,W.
  7    FTN,I=DOJOB,L=0.
  8    MAP,OFF.
  9    MODE,0
 10    LGO.
 11    EOR (stuurkaarten-jobcontrole)
 12    *COMDECK DOTHIS
 13          P1=300.
 14          P1=500.
 15          P2= .50
 16          P2= .55
 17          P2= .60
 18    EOR (eerste update, creation run)
 19    EOR (tweede update, merge run)
 20    *IDENT NEW
 21    *DELETE MAIN.201
 22          DO 22 I=1,N
 23    *INSERT THEORY.16
 24    C     BEREKENING XVALUE
 25    EOR (derde update, correction run, compile)
 26    EOF (job)

-------1234567890 = kaartkolom
```

Voorbeeld 3.3. Volledige computerjob voor het creëren van een Update-NPL van de door het hoofdprogramma MAIN op te roepen common-deck DOTHIS. Dit comdeck bevat de toe te passen alternatieve parameterwaarden voor P1 en P2 die het hoofdprogramma completeren. Het comdeck wordt hiermede op één NPL gezet namelijk NEWJOB. Hierna wordt het hoofdprogramma nog verbeterd, wordt vertaald en uitgevoerd (zie toelichting hiernaast).

Regel Toelichting bij Voorbeeld 3.3
(zie ook fig. 1, pag. 33)

1 t/m 3 Overeenkomstig als in JOB2, de te reserveren geheugenruimte is 70000 en daarmee aangepast aan de ruimte benodigd voor het vertalen van het programma

4 Het creëren van een NPL met file naam DOWHAT. De te creëren update file wordt van de Input file gelezen van EOR tot EOR en bestaat dan dus uit het comdeck DOTHIS

5 De in 4 gecreëerde NPL-file DOWHAT heeft nu de status van OPL-file. Deze file wordt samen met de update-file CASE (gecreëerd in de tweede job door het 'mergen' van de files MAIN en CDS) gemengd tot één update-file (merge met: M=DOWHAT) en de nieuwe update-file wordt nu NEWJOB genoemd. Er wordt nog niets naar de compile-file geschreven. (De ervaring leert dat dit beter niet tegelijk met een 'merge' kan plaatsvinden)

6 De NPL uit 5 is thans weer een OPL, namelijk P=NEWJOB. Er wordt verondersteld dat in het hoofdprogramma tevens nog een verbetering moet worden aangebracht (regel 22) en dat in het gedeelte aangeduid met THEORY een comment-kaart moet worden toegevoegd (regel 24). De Input voor deze Update-correctie run loopt van EOR tot EOR (regel 19 tot 25). Alle onderdelen worden nu naar de compile-file geschreven (C=DOJOB) en tot een volledig programma samengesteld. De output van deze update bevat nu de volgende mededelingen:

```
COMMON DECKS ENCOUNTERED
DOTHIS      TEXT        THEORY

DECKS WRITTEN TO COMPILE FILE
MAIN
```

Zonder de F (full mode) optie verschijnt deze mededeling niet en wordt er geen compile file gemaakt.

7 Het Fortran-programma, voorkomend op de Input-file DOJOB wordt vertaald. Er wordt geen programma list geprint (L=0). Een volledige list van de Fortran-tekst inclusief de volledige programma organisatie wordt verkregen met R=3, in plaats van L=0

8 Er wordt geen overzicht van de geheugen inhoud gegeven

9 Er wordt niet gestopt bij numeriek onmogelijke uitkomsten zoals van $\frac{1}{0}$ of $\sqrt{-3}$. (Deze optie kan juist nuttig zijn om dergelijke fouten op te sporen)

10 Load and Go: het programma wordt uitgevoerd.

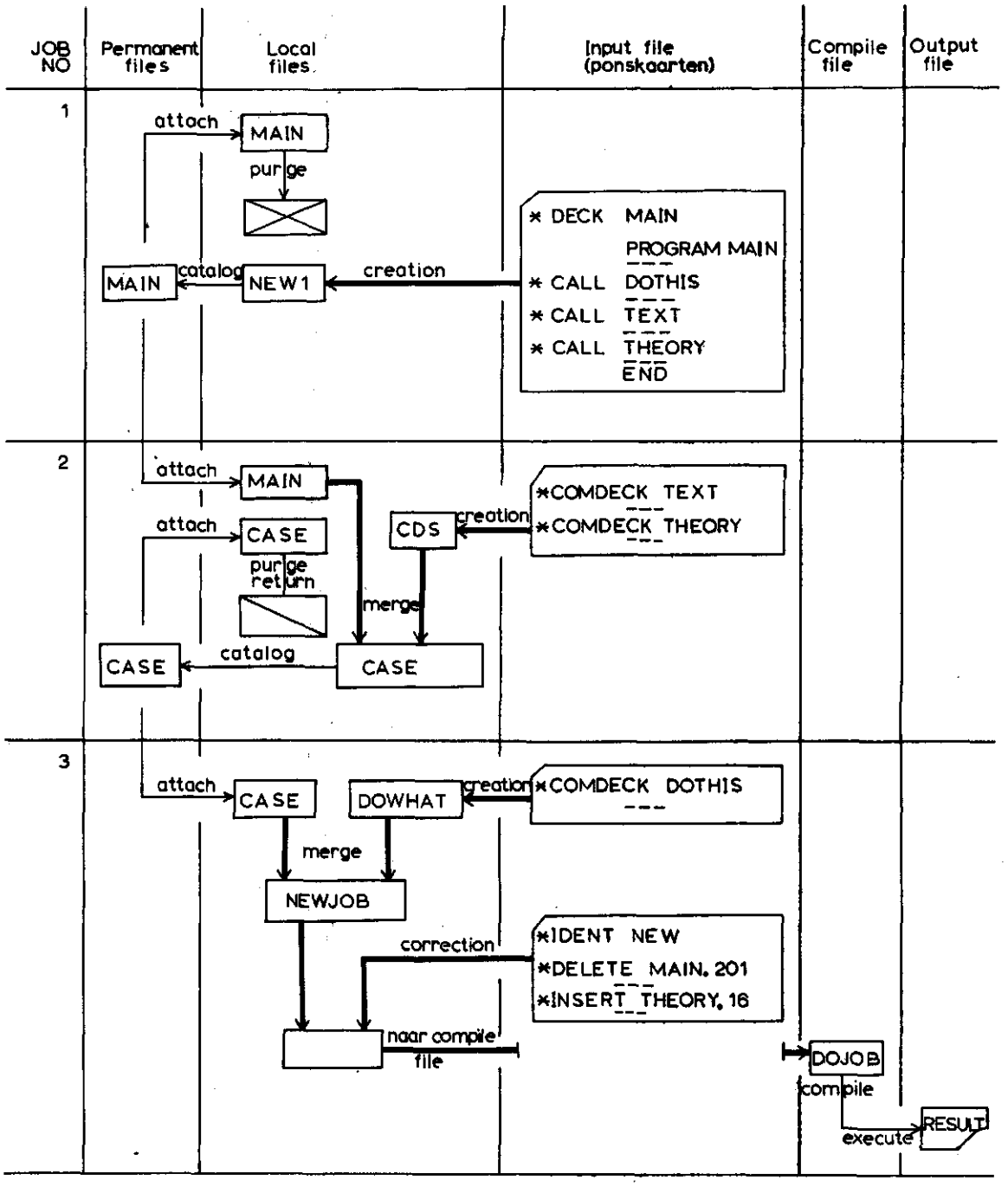


Fig.1 Illustratie van file flow in JOB 1, 2 en 3 uit voorbeeld 3.1; 3.2; 3.3

9.1.2. Opmerkingen bij voorbeeld 3.3

De Update-nummering vindt plaats volgens deck-naam, comdeck-naam en identifier-naam van respectievelijk creation- en correction runs. Teneinde een indruk te geven hoe de nummering thans verloopt, wordt de volgende schets gegeven:

```
      PROGRAM MAIN(INPUT...)      MAIN     2
      ---                         MAIN     3
      .                           .
      .                           .
      .                           .
      ---                         MAIN    24
      P1=300.                     DOTHIS   2
      P1=500.                     DOTHIS   3
      P2=.50                      DOTHIS   4
      P2=.55                      DOTHIS   5
      P2=.60                      DOTHIS   6
      ---                         MAIN    26
      .                           .
      .                           .
      .                           .
      ---                         MAIN   200
      DO 22 I=1,N                 NEW      1
      ---                         MAIN   202
      .                           .
      .                           .
      .                           .
      ---                         THEORY  16
C     BEREKENING XVALUE           NEW      2
      ---                         THEORY  17
      .                           .
      .                           .
      .                           .
      etc.
```

Tenslotte wordt verwezen naar fig. 1 waarop de file-flow van de drie jobs in hun onderlinge samenhang staat weergegeven.

Opgemerkt wordt dat het gegeven schema niet op dezelfde wijze als een stroomdiagram voor een Fortrantekst kan worden gelezen: er is niét een enkel beginpunt en een eenduidige volgorde, Zo zullen bijvoorbeeld in de tweede job de files MAIN en CASE achtereenvolgens ge-ATTACH-ed kunnen worden, doch de 'attach' van MAIN kan ook plaatsvinden vlak voor de UPDATE-instructie waarbij deze file met de file CDS gemengd wordt. Ook kan men met een bepaalde file verschillende handelingen verrichten zodat de aanduiding van die file meer dan één uitgang kan hebben.

De bedoeling van de figuur komt het best tot uiting door de figuur te lezen aan de hand van de corresponderende jobs.

9.1.3. Voor- en nadelen bij voorbeeld 3

Omtrent het hier gegeven systeem kunnen de volgende voor- en nadelen worden aangegeven:

+) Het hoofdprogramma staat op permanent file en er behoeft niet met het kaartendek hiervan gemanipuleerd te worden

+) De alternatieve functie, opgenomen in comdecks TEXT en THEORY, kan met behulp van de routine job JOB2 eenvoudig in het hoofdprogramma worden ingebouwd; voor ieder nieuw in te bouwen functie behoeven slechts de Fortran teksten in de comdecks TEXT en THEORY vervangen te worden: dezelfde JOB2 kan worden gebruikt

+) Voor het doorrekenen met alternatieve parameterwaarden behoeft slechts rekening gehouden te worden met de inhoud van het comdeck DOTHIS. De kaarten bevatten ook de naam van de parameters wat de overzichtelijkheid ten goede komt (voorbeeld 3,3 regel 13 t/m 17)

+) Er kunnen tevens aanvullende wijzigingen of verbeteringen in het programma worden aangebracht door één of meer instructies te wijzigen (voorbeeld 3.3 regel 20-24), zonder met het gehele kaartendek van het hoofdprogramma te hoeven te manipuleren

-) Het hoofdprogramma moet voor elke berekening opnieuw worden vertaald

-) Er moet rekening mee worden gehouden dat in regel 6 van voorbeeld 3.3 geen NPL wordt gemaakt zodat de verbeteringen onder *IDENT NEW aangebracht niet permanent in de UPDATE-versie zijn aangebracht

Voor een overzicht over de jobroulatie wordt verwezen naar hoofdstuk 10.

9.2. Voorbeeld 4: U P D A T E - e n z o n d e r h e r h a a l d v e r t a l e n v a n h e t h o o f d p r o - g r a m m a

Het laatstgenoemde nadeel kan worden ondervangen door het vertaalde hoofdprogramma eveneens op een permanent file te plaatsen en deze op te roepen in de volgende jobs. Het is dan echter niet mogelijk op dezelfde wijze gedeelten van het programma uitwisselbaar te maken. Het vertaalde programma is echter wel in staat een subroutine op te roepen zodat onder gebruikmaking van deze mogelijkheid de uitwisselbaarheid behouden kan blijven.

Als nadeel kan nu gelden dat meer file-administratie moet worden verzorgd: ook de vertaalde - de zogenaamde binaire - file moet roulerend in de jobverwerking worden opgenomen. Tevens zal het aanbrengen van kleine veranderingen in het hoofdprogramma door middel van een update-correction run in een aparte job moeten plaatsvinden aangezien juist deze handeling weer een nieuwe vertaling van het hoofdprogramma vraagt.

In het laatste voorbeeld zal aangegeven worden hoe in dit geval de job-opbouw zou kunnen luiden en hoe de jobs op elkaar kunnen aansluiten.

Aangezien nu de te variëren parameters in een subroutine zullen worden gedefinieerd moet er een koppeling tot stand komen tussen variabelen gedefinieerd in een subprogramma en dezelfde variabelen zoals die gebruikt worden in het hoofdprogramma. Dit vindt plaats met een COMMON instructie die ervoor zorgt dat de genoemde variabelen op dezelfde geheugenplaats worden opgeslagen. Voorbeeld:

```
COMMON P1,P2
```

een instructie welke zowel in het hoofdprogramma als in de subroutine moet voorkomen.

Verondersteld wordt nu dat de te variëren parameterwaarden zijn opgenomen in een Fortran subroutine (zie JOP3):

```
SUBROUTINE DOTHIS
COMMON P1,P2
P1=300.
P1=500.
P2=.50
P2=.55
P2=.60
END
```

Ook nu is het manipuleren met het kaartendek eenvoudig. Op dezelfde wijze als voorheen omschreven kan een cyclische verwisseling van kaarten worden toegepast om in achtereenvolgende jobs de verschillende parametercombinaties te verkrijgen.

Het volledige voorbeeld bestaat uit 5 jobs, de opbouw waarvan vergeleken kan worden met fig. 2 op pag. 49.

9.2.1. Naamgeving alternatieve functies in Update

Er wordt nog de volgende verfijning aangebracht.

Het hoofdprogramma bevat de instructie *CALL TEXT in welk comdeck verondersteld werd dat daarin enige begeleidende en toelichtende tekst op de in behandeling zijnde modificatie van de functie wordt gegeven. Zou men een alternatieve functie gaan gebruiken, dan moet toch het comdeck de naam TEXT blijven houden aangezien in het hoofdprogramma deze naam gebruikt wordt en anders steeds het hoofdprogramma zou moeten worden aangepast. Onder toepassing van een leeg comdeck kan men bereiken dat zowel het kaartendek zelf eenduidig kan worden benoemd, en dat tevens in de output van de UPDATE deze benaming vermeld wordt evenals in de DECK LIST van de ITEMIZE, zodat steeds is na te gaan welk alternatief op een bepaald moment in het hoofdprogramma is ingebouwd.

Aangezien een comdecknaam uit 9 tekens mag bestaan (inclusief enkele bijzondere tekens als /, - * en +) kan meestal voor een duidelijke identificatie worden gezorgd (zie STOL, 1977a).

Voorbeeld:

```
*COMDECK FORMULE/1
*COMDECK TEXT
C
C   EERSTE METHODE
C
C   BEREKENING MET FORMULE 1
C
      WRITE(3,15)
   15 FORMAT(1H1"FORMULE 1"//)
*COMDECK THEORY
      -
      -
      - fortran-tekst formule 1
      -
      -
```

en als alternatief:

```
*COMDECK FORMULE/2
*COMDECK TEXT
C
C   TWEEDE METHODE
C
C   BEREKENING MET  FORMULE 2
C
      WRITE(3,15)
   15 FORMAT(1H1"FORMULE 2"//)
*COMDECK THEORY
      -
      -
      - fortran-tekst formule 2
      -
      -
```

Met het UPDATE-systeem is het op een eenvoudige wijze mogelijk de job-opbouw zo te kiezen dat gehele comdecks verwijderd worden uit de OPL en de NPL opgebouwd wordt met de nieuwe comdecks (zie JOP5).

Gewezen kan nog worden op het voordeel dat elk kaartendek een identificatie heeft gekregen, namelijk FORMULE/1 en FORMULE/2, wat het archiveren ten goede komt. Wel moet men nog beschikken over de stuurkaarten *PURGE FORMULE/2 respectievelijk *PURGE FORMULE/1 om de alternatieve functies onderling te kunnen uitwisselen (zie ook hiervoor JOP5).

Dezelfde identificatie van de alternatieve formules komt voor in de output van de Update- en Itemize-instructies hetgeen de overzichtelijkheid bevordert. Een Itemize van de file CASE geeft als belangrijkste output voor deze update-file:

```
ITEMIZE OF CASE
DECK LIST.
        MAIN        FORMULE/1   TEXT        THEORY
```

of, na inzetten van het alternatief volgens JOP5;

```
ITEMIZE OF CASE
DECK LIST.
        MAIN        FORMULE/2   TEXT        THEORY
```

waaruit het voordeel van het gebruik van een definiërende - overigens lege en niet door het hoofdprogramma opgeroepen - comdeck blijkt.

Voor voorbeeld 4.1 zie volgende pagina

```
Regel--1234567890 = kaartkolom

  1    JOP1,CL50000,T10,IO10.
  2    ACCOUNT,...
  3    ATTACH,MAIN,MAIN,ID=...
  4    PURGE,MAIN.
  5    REQUEST,NEW1,*PF.
  6    UPDATE,N=NEW1,C=0,W,L=0.
  7    ITEMIZE,NEW1,N.
  8    CATALOG,NEW1,MAIN,ID=...,MR=1.
  9    EOR(stuurkaarten-jobcontrol)
 10    *DECK MAIN
 11          PROGRAM MAIN(INPUT,...)
 12          COMMON P1,P2
 13          fortran tekst hoofdprogramma
 14          CALL DOTHIS
 15          vervolg fortran tekst hoofdprogramma
 16    *CALL TEXT
 17          vervolg fortran tekst hoofdprogramma
 18    *CALL THEORY
             vervolg fortran tekst hoofdprogramma
 20          END
 21    EOR (eerste update, creation run)
 22    EOF (job)

-------1234567890 = kaartkolom
```

Voorbeeld 4.1. Volledige computerjob voor het creëren van een Update-NPL van het hoofdprogramma MAIN, ondergebracht in het Update-deck MAIN. In de Fortran tekst van het programma komen oproepen voor van één subroutine en van twee nader te vullen Update-comdecks. De comdecks bevatten de Fortran teksten die het hoofdprogramma moeten gaan completeren (zie toelichting hiernaast).

Regel	Toelichting bij Voorbeeld 4.1 (zie ook fig. 2, pag. 49)
1 t/m 9	De job-opbouw is gelijk aan die welke in voorbeeld 3.1 werd gegeven
12	Het hoofdprogramma bevat een COMMON-kaart om de overdracht van waarden, gedefinieerd in de subroutine, te bewerkstelligen
14	De *CALL DOTHIS (een UPDATE-instructie!!) is vervangen door een CALL DOTHIS (een Fortran-instructie!!)
	Voor details zie Toelichting bij Voorbeeld 3.1.

```
Regel--1234567890 = kaartkolom

  1    JOP2,CL70000,T10,IO10.
  2    ACCOUNT,...
  3    ATTACH,CASE,CASE,ID=...
  4    PURGE,CASE.
  5    RETURN,CASE.
  6    ATTACH,CASEB,CASEB,ID=...
  7    PURGE,CASEB.
  8    RETURN,CASEB.
  9    REQUEST,CASE,*PF.
 10    REQUEST,CASEB,*PF.
 11    ATTACH,MAIN,MAIN,ID=...
 12    UPDATE,N=CDS,C=0,L=F,W.
 13    UPDATE,P=MAIN,M=CDS,N=CASE,C=0,W.
 14    UPDATE,P=CASE,C=DOJOB,F,W.
 15    FTN,I=DOJOB,B=CASEB,R=3.
 16    CATALOG,CASE,CASE,ID=...,MR=1
 17    CATALOG,CASEB,CASEB,ID=...,MR=1.
 18    EOR (stuurkaarten-jobcontrol)
 19    *COMDECK FORMULE/1
 20    *COMDECK TEXT
 21          fortran tekst m.b.t. formule 1
 22    *COMDECK THEORY
 23          fortran tekst m.b.t. formule 1
 24    EOR (eerste update, creation run)
 25    EOR (tweede update, merge run)
 26    EOR (derde update, compile file)
 27    EOF (job)

-------1234567890 = kaartkolom
```

Voorbeeld 4.2. Volledige computerjob voor het creëren van een Update-NPL van de door het hoofdprogramma MAIN op te roepen common-decks TEXT en THEORY. De comdecks bevatten de Fortran-teksten die het hoofdprogramma completeren en worden hiermede op één NPL (N=CASE) gezet. Hierna wordt het hoofdprogramma vertaald (zie toelichting hiernaast).

Regel	Toelichting bij Voorbeeld 4.2 (zie ook fig. 2, pag. 49)
1-13	Verloopt analoog aan JOB2. Er komt echter een extra file voor waarop het binaire vertaalde Fortranprogramma geplaatst wordt (CASEB)
14	De met de comdeck's TEXT en THEORY vermengde file MAIN wordt naar de compile file geschreven (C=DOJOB). De UPDATE-instructie bevat de parameter F (full mode) teneinde te bereiken dat alle comdecks op de compile file zullen voorkomen
15	Het programma dat op de inputfile I=DOJOB staat wordt vertaald en geplaatst op de binaire file B=CASEB
16	Update file CASE wordt gecatalogiseerd
17	Binaire file CASEB wordt gecatalogiseerd
19	Een leeg comdeck dat alleen dient om een aanduiding omtrent de alternatieve formule in het systeem te bewaren

```
Regel--1234567890 = kaartkolom

  1     JOP3,CL70000,T20,IO10.
  2     ACCOUNT,...
  3     ATTACH,CASEB,CASEB,ID=...
  4     FTN,B=DOWHAT,L=0.
  5     MAP,OFF.
  6     MODE,0.
  7     LOAD,DOWHAT
  8     LOAD,CASEB.
  9     EXECUTE.
 10     EOR (stuurkaarten-jobcontrol)
 11           SUBROUTINE DOTHIS
 12           COMMON P1,P2
 13           P1=300.
 14           P1=500.
 15           P2= .50
 16           P2= .55
 17           P2= .60
 18           END
 19     EOR (fortran programma)
 20     EOF (job)

-------1234567890 = kaartkolom
```

Voorbeeld 4.3. Volledige computerjob voor het vertalen van de subroutine DOTHIS. Deze subroutine bevat de toe te passen alternatieve parameterwaarden voor P1 en P2 die het hoofdprogramma completeren. De binaire files die de vertaalde programma's bevatten worden geladen en uitgevoerd (zie toelichting hiernaast).

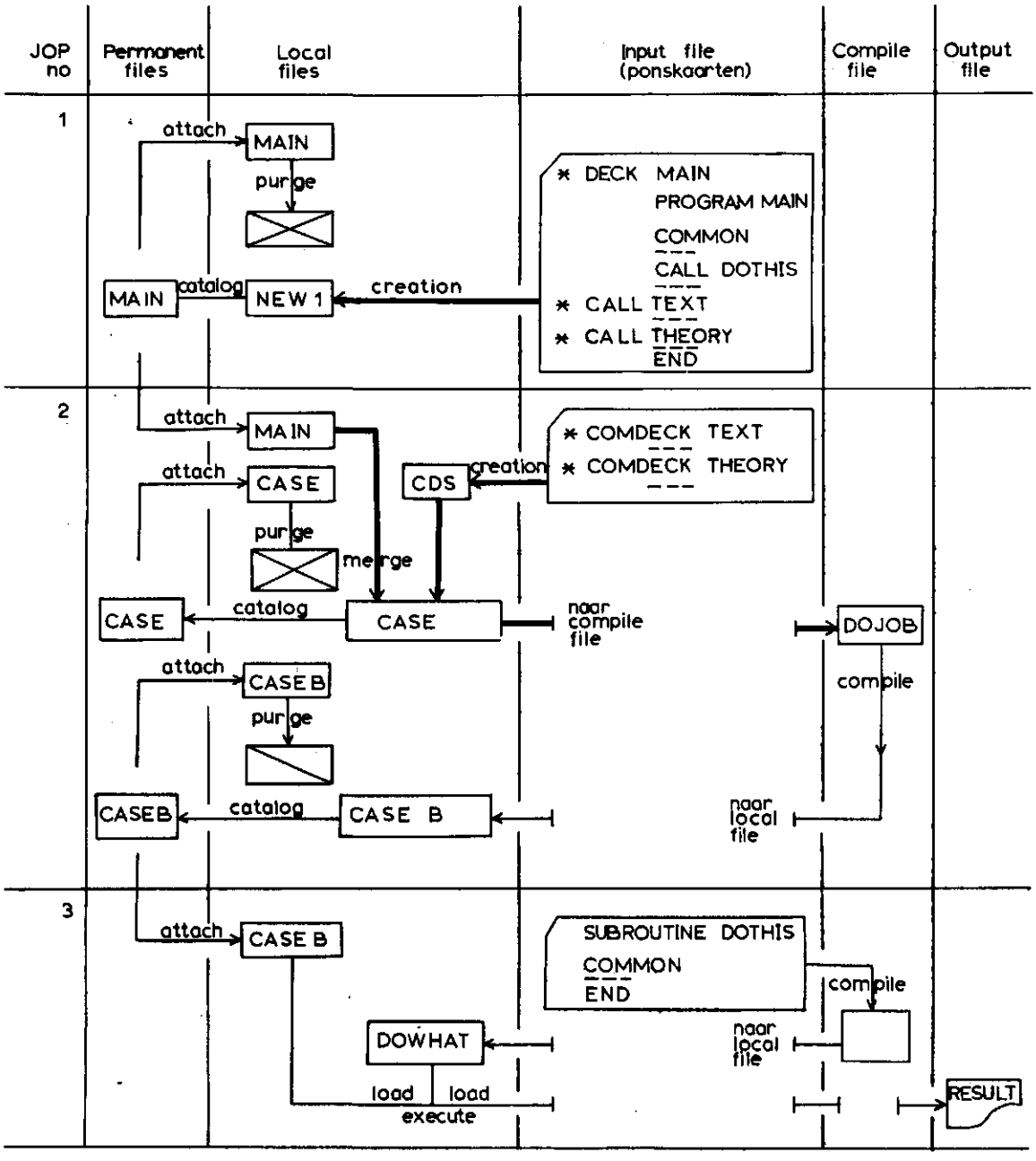
Regel	Toelichting bij Voorbeeld 4.3 (zie ook fig. 2, pag. 49)
1,2	Overeenkomstig JOB3
3	De permanent file CASEB, die het vertaalde hoofdprogramma bevat wordt local gemaakt en op regel 8 geladen
4	De Subroutine DOTHIS wordt vertaald, het vertaalde programma wordt op de binaire file B=DOWHAT geplaatst en op regel 7 geladen
5,6	Als in JOB3
7,8	Laden van de vertaalde binaire Fortran programma's van respectievelijk de subroutine en het hoofdprogramma
9	De vertaalde programma's worden uitgevoerd
13,17	Kaarten met parameterwaarden die verwisseld kunnen worden om de juiste combinatie (de laatste) toe te passen

9.2.2. Opmerkingen bij voorbeeld 4.3

Het nu ontworpen systeem werkt op exact dezelfde wijze als dat uit voorbeeld 3. Het verschil in opbouw is echter dat de parameterwaarden in een subroutine en dus niet in het hoofdprogramma worden gedefinieerd. Teneinde een indruk te geven hoe de nummering thans verloopt, wordt de volgende schets gegeven. De met Update uit te voeren verbeteringen en aanvullingen (zie JOP4) zijn nog niet aangebracht gedacht:

```
PROGRAM MAIN(INPUT...)      MAIN    2
---                         MAIN    3
.
.
.
---                         MAIN   24
CALL DOTHIS                 MAIN   25
---                         MAIN   26
.
.
.
---                         MAIN  200
DO 22 I=1,M                 MAIN  201
---                         MAIN  202
.
.
.
---                         THEORY 16
---                         THEORY 17
.
.
.
etc.
```

De Subroutine DOTHIS is niet op een Update file geplaatst en heeft dus geen update nummering.



Legenda: zie fig. 1

Fig. 2 Illustratie van file flow in JOP 1, 2 en 3 uit voorbeeld 4.1 4.2 en 4.3

Tenslotte wordt verwezen naar fig. 2 waarop de file-flow van de drie jobs in hun onderlinge samenhang staat weergegeven.

Ook deze figuur kan het best gezamenlijk met de corresponderende jobs worden beschouwd.

Voor voorbeeld 4.4 zie volgende pagina.

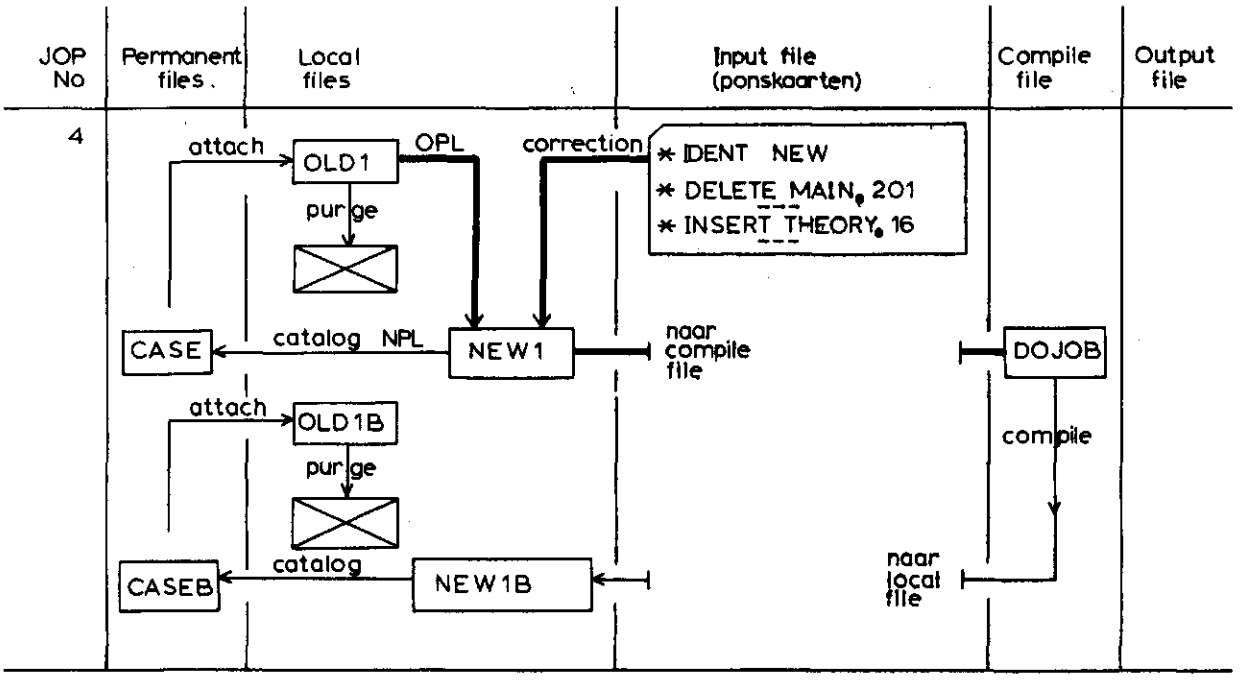
```
Regel--1234567890 = kaartkolom

  1    JOP4,CL70000,T20,IO10.
  2    ACCOUNT,...
  3    ATTACH,OLD1,CASE,ID=...
  4    ATTACH,OLD1B,CASEB,ID=...
  5    UPDATE,P=OLD1,N=NEW1,C=DOJOB,F,L=0,W.
  6    FTN,I=DOJOB,B=NEW1B,L=0.
  7    PURGE,OLD1.
  8    PURGE,OLD1B.
  9    CATALOG,NEW1,CASE,ID=...,MR=1.
 10    CATALOG,NEW1B,CASEB,ID=...,MR=1.
 11    EOR (stuurkaarten-jobcontrol)
 12    *IDENT NEW
 13    *DELETE MAIN.201
 14          DO 22 I=1,N
 15    *INSERT THEORY.16
 16    C     BEREKENING XVALUE
 17    EOR (eerste update, correction, compile)
 18    EOF (job)

-------1234567890 = kaartkolom
```

Voorbeeld 4.4. Volledige computerjob voor het updaten van het hoofdprogramma dat de inhoud vormt van file CASE. De uit te voeren verbeteringen komen voor tussen de EOR-kaarten 11 en 17 (zie toelichting hiernaast).

Regel	Toelichting bij Voorbeeld 4.4 (zie ook fig. 3, pag. 55)
1,2	Als in JOP2
3	De update-file CASE wordt local gemaakt
4	De binaire file CASEB wordt local gemaakt
	Beide files zijn van belang. De update-file wordt verbeterd, daarna vindt een nieuwe vertaling plaats zodat de binaire file ook vervangen moet worden
	Voor de overige onderdelen van deze job wordt verwezen naar JOB3 in voorbeeld 3.3 waarin dezelfde verbeteringen worden aangebracht
7,8	De oude files worden verwijderd
9,10	De nieuwe files worden op permanent file gezet

9.2.3. Opmerkingen bij voorbeeld 4.4

In vorige voorbeelden werden files die vervangen moesten worden na de attach gepurged zoals in JOB1, voorbeeld 3.1. Het kan van voordeel zijn het purgen uit te stellen totdat de nieuwe file is aangemaakt. Zou hierin een fout voorkomen dan breekt de job vaak automatisch af en gaan er geen files verloren. Het gehele verwerkingssysteem blijft dan roulerend. Dezelfde job kan na te zijn verbeterd, zonder meer opnieuw worden ingelezen en verwerkt daar dan alle benodigde files nog in het systeem aanwezig zijn.



Legenda: zie fig. 1

Fig. 3 Illustratie van file flow in JOP 4

```
Regel--1234567890 = kaartkolom

  1     JOP5,CL70000,T20,IO10.
  2     ACCOUNT,...
  3     ATTACH,OLD1,CASE,ID=...
  4     ATTACH,OLD1B,CASEB,ID=...
  5     UPDATE,P=OLD1,N=PURGE,C=0,W.
  6     UPDATE,N=CDS,C=0,L=F,W.
  7     UPDATE,P=PURGE,M=CDS,N=NEW1,C=0,W.
  8     UPDATE,P=NEW1,C=DOJOB,F,L=0,W.
  9     FTN,I=DOJOB,B=NEW1B,R=3.
 10     ITEMIZE,NEW1,N.
 11     ITEMIZE,NEW1B,N.
 12     PURGE,OLD1.
 13     PURGE,OLD1B.
 14     CATALOG,NEW1,CASE,ID=...,MR=1.
 15     CATALOG,NEW1B,CASEB,ID=...,MR=1.
 16     EOR (stuurkaarten-jobcontrol)
 17     *PURGE FORMULE/1
 18     *PURGE TEXT
 19     *PURGE THEORY
 20     EOR (eerste update, purge run)
 21     *COMDECK FORMULE/2
 22     *COMDECK TEXT
 23           fortran tekst m.b.t. formule 2
 24     *COMDECK THEORY
 25           fortran tekst m.b.t. formule 2
 26     EOR (tweede update, creation run)
 27     EOR (derde update, merge run)
 28     EOR (vierde update, compile file)
 29     EOF (job)

-------1234567890 = kaartkolom
```

Voorbeeld 4.5. Volledige computerjob voor het werwijderen van alle comdecks die betrekking hebben op formule 1 en het creëren van de NPL van alle comdecks die betrekking hebben op formule 2. De nieuwe comdecks nemen aan het eind van de job de plaats van de oude in (zie toelichting hiernaast)

Regel Toelichting bij Voorbeeld 4.5
(zie ook fig. 4, pag. 59)

Na de toelichtingen op de vorige jobs behoeft deze job weinig toelichtend commentaar. De nieuwe elementen zijn:

5 Van de OPL (P=OLD1) worden volgens de uit te voeren update-instructies (regel 17 t/m 20) de comdecks FORMULE/1, CASE en THEORY verwijderd. De nieuwe update-file heet nu N=PURGE (een zelf verzonnen naam overigens)

6 De comdecks, gedefinieerd in de regels 21 t/m 26, werden op de update-file N=CDS geplaatst met een creation run

7 De file CDS wordt gemengd (gemerged) met de file PURGE en de NPL heet nu NEW1

8 De file NEW1 waarop nu de nieuwe comdecks staan geschreven worden naar de compile file gecopieerd. De parameter F (full mode) draagt er zorg voor dat alle comdecks op de compile file worden geplaatst zodat het Fortran-programma kan worden gecompleteerd

Voor overige stuurkaarten: zie toelichting bij vorige jobs.

9.2.4. Voor- en nadelen bij voorbeeld 4

Omtrent dit laatste systeem kunnen nog de volgende voor- en nadelen worden aangegeven:

+) Het hoofdprogramma staat als vertaalde binaire file op permanent file en er behoeft niet met het kaartendek hiervan gemanipuleerd te worden

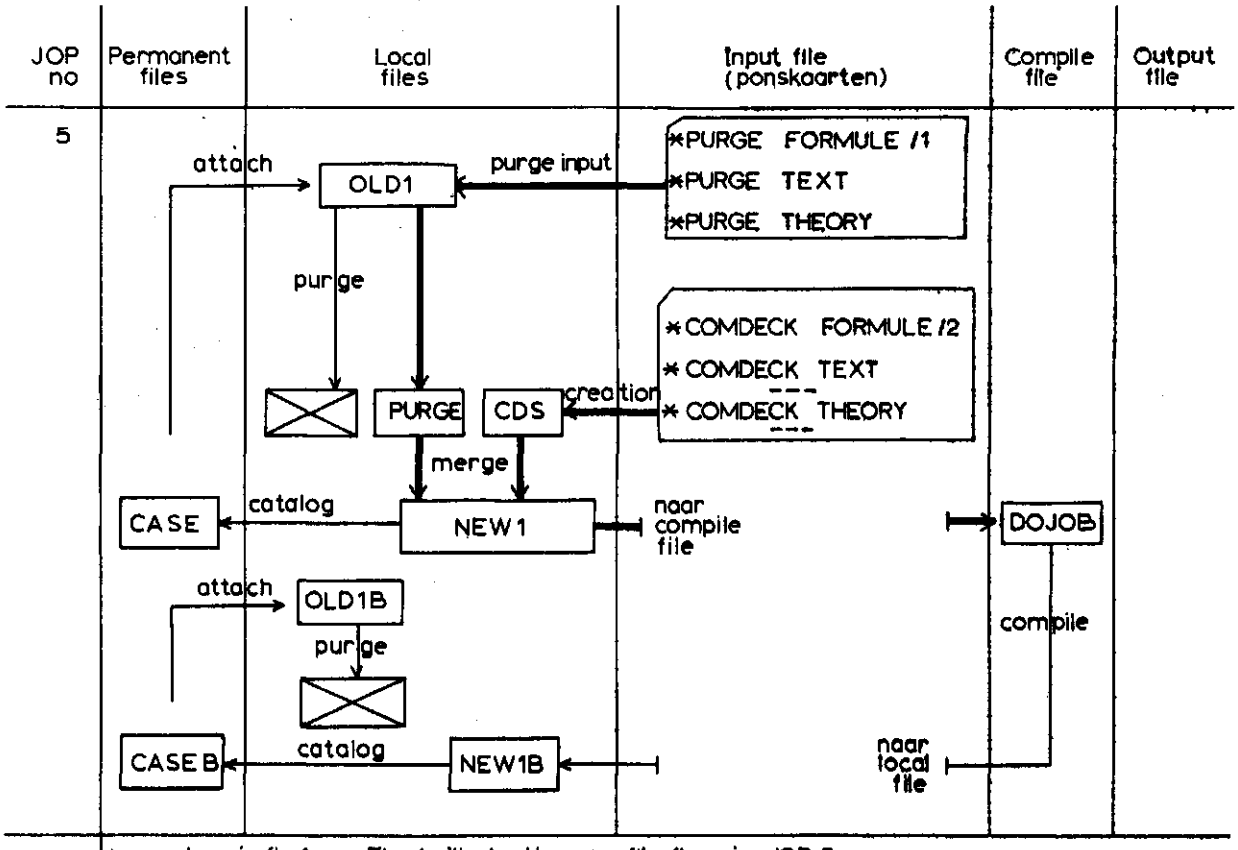
+) Het hoofdprogramma behoeft voor elke alternatieve parameterwaarden combinatie niet weer opnieuw vertaald te worden

+) Kaartendekjes met alternatieve formules kunnen apart bewaard worden en wanneer nodig direct in een update-file de oude versie vervangen in een purge-job

+) Voor het doorrekenen met alternatieve parameterwaarden behoeft slechts rekening gehouden te worden met de inhoud van de subroutine DOTHIS in JOP3. De kaarten bevatten ook de naam van de parameters wat de overzichtelijkheid ten goede komt (voorbeeld 4.3 regel 13 t/m 17)

-) De file administratie is ingewikkelder geworden. Worden de jobs opgebouwd zoals in het voorbeeld is aangegeven dan kan men zijn aandacht echter verder geheel op het hoofdprogramma, de alternatieve functies en de vereiste combinatie van parameterwaarden richten

Voor een overzicht over de jobroulatie wordt verwezen naar hoofdstuk 9.



Legenda: zie fig.1 Fig. 4 Illustratie van file flow in JOP 5

10. JOBROULATIE

Door de namen van de files dusdanig te kiezen dat de jobs in voorbeeld 3 (en 4) onderling op elkaar aansluiten zal het niet nodig zijn verdere zorg aan de jobopbouw te besteden. De uitzonderingen zijn:

- Bij de allereerste run bestaan de te gebruiken files nog niet en moeten de ATTACH, PURGE en RETURN kaarten worden verwijderd
- Afhankelijk van de wens of men een programmalijst wil (R=3) of niet (L=0) moet de FTN-kaart worden aangepast
- Veelal is voor de update bewerking geen output (L=0) gekozen. Eventuele fouten worden wel gemeld. De default output die het verloop van de update zichtbaar maakt verkrijgt men door de parameter L niet te definiëren en weg te laten
- Bij het verloren gaan van files zal nagegaan moeten worden met welke voorgaande job deze hersteld kunnen worden. Nagegaan moet tevens worden of de ATTACH kaarten gebruikt kunnen worden

De jobs kunnen achter elkaar worden toegepast en wel als volgt:

Taak	Herhaald vertalen	
	met	zonder
Hoofdprogramma in update	JOB1	JOP1
Toevoegen comdecks	JOB2	JOP2
Alternatieve parameters	JOB3	JOP3
Verbeteren programma	JOB3	JOP4
Weer doorrekenen	n.v.t.	JOP3
Alternatieve functie inzetten	JOB2	JOP5
Weer doorrekenen	JOB3	JOP3

11. AANVULLENDE OPMERKINGEN OVER JOBOPBOUW

Enkele detail-opmerkingen over de jobopbouw dienen te worden gemaakt.

. Afhankelijk van het systeem waarop gewerkt wordt dient de jobnaam aan bepaalde administratieve voorwaarden te voldoen. Overleg met de afd. Wiskunde is hiervoor noodzakelijk.

. De gekozen verwerkingstijd (T) en in- en outputtijd (IO) zal veelal voldoende zijn. Afhankelijk van de opgedane ervaring met een gegeven functie kan men de tijden opnieuw, en zo zuinig mogelijk, kiezen: de tijden bepalen mede de prioriteit van de job.

. De benodigde geheugenruimte is voor alle Update-jobs ruim voldoende gekozen. Voor zeer grote programma's zal er meer geheugenruimte voor de feitelijke verwerking gereserveerd moeten worden; voor het vertalen alleen is de opgegeven waarde voldoende.

. Het gebruik van de MODE,0.-kaart kan van voordeel zijn voor het opsporen van fouten in de numerieke uitwerking. Er wordt dan namelijk niet gestopt bij de bekende onmogelijke operaties als $\frac{1}{0}$ en $\sqrt{-3.}$ Het gevaar is echter dat de bewerking hierdoor onevenredig veel tijd kan vergen, en de job daardoor duur wordt, of dat er veel onzinnige output verschijnt. Een juiste afweging van het gebruik van deze kaart tegen de opgegeven verwerkingstijden is steeds noodzakelijk.

. Zonder PURGE kan wel een localfile naar een permanent file met reeds gebruikte naam worden overgebracht.De local file komt dan op CYCLE 002 etc. Er zal dan de benodigde administratie moeten worden gevoerd met betrekking tot op welke cycle welke versie staat en met betrekking tot het geven van de juiste ATTACH-instructie.

. Hoewel het aanbevolen wordt een REQUEST, local filename, *PF-instructie te gebruiken alvorens een CATALOG naar een permanent file uit te voeren is in de praktijk de noodzaak ervan niet gebleken.

In enkele voorbeelden is dan ook niet van de REQUEST gebruik gemaakt om dit aan te tonen.

. Wordt een REQUEST-instructie gebruikt, dan wordt dat in het systeem als definitie van een nieuw toe te voegen file opgevat. Bij een roulerend systeem zal aan de REQUEST vooraf een RETURN moeten zijn gegeven (zie voorbeeld 3.2, regel 4,5 en 6).

. Heeft men, als voorbeeld, aangemaakt de permanent file TESTJOB, dan zijn de resultaten van de volgende instructies:

```
ATTACH,TEST,TESTJOB.
PURGE,TEST.
UPDATE,N=TEST.
```

Correct: na de PURGE kan de local file naam opnieuw worden gebruikt.

```
ATTACH,TEST,TESTJOB.
UPDATE,N=TEST.
```

Niet correct: de local file naam TEST is reeds in het systeem aanwezig. Er verschijnt in de dayfile de boodschap ILLEGAL I/O REQUEST.

```
ATTACH,TEST,TESTJOB.
PURGE,TEST.
UPDATE,P=TEST,N=NEWTEST.
CATALOG,NEWTEST,TESTJOB.
```

Correct: de file TEST is nog in het systeem aanwezig.

```
ATTACH,TEST,TESTJOB.
PURGE,TEST.
RETURN,TEST.
UPDATE,P=TEST.
```

Niet correct: de file TEST komt niet meer in het systeem voor. Er verschijnt de boodschap: NO OPL.

```
ATTACH,TEST,TESTJOB.
PURGE,TEST.
REQUEST,TEST,*PF.
```

Niet correct zonder aan de REQUEST voorafgaande RETURN,TEST. De local file naam komt nog in het systeem voor. Er verschijnt in de dayfile de boodschap: DUPLICATE FILE NAME, REQUEST ABORTED.

. Voor en na een ITEMIZE behoeft geen REWIND te worden gegeven, evenmin als voor en na CATALOG.

. Met *PURGE deckname verdwijnen ook de verbeteringen door middel van een update correction run onder elke opgegeven *IDENT aangebracht. De, onder dezelfde *IDENT, aangebrachte verbeteringen in andere programma gedeelten blijven bestaan.

. Behoeven er geen verbeteringen te worden aangebracht, dan mag het update input record leeg zijn (d.w.z. de regels 20 t/m 24 in voorbeeld 3.3, JOB3, kunnen weggelaten worden) de EOR-kaarten blijven noodzakelijk.

. Bij het rouleren van jobs waarin onder een opgegeven *IDENT-naam verbeteringen zijn aangebracht moet bedacht worden dat bij de volgende update-correction run een nieuwe *IDENT-naam moet worden gebruikt. Wordt dit niet gedaan, dan worden alle correctie-instructies gemerkt met *ERROR*. Er volgt geen update. Dit kan ervoor pleiten de verbeterde file eerst te catalogiseren als álle verbeteringen juist zijn uitgevoerd.

LITERATUUR

STOL, PH.TH., 1977. Over het samenstellen van een computerprogramma voor het optimaliseren van parameters. Aspecten van Informatieverwerking no. 6; ICW-Nota 951

_____ 1977a. Optimaliseren van parameters. Het gereedmaken van een functie voor toepassing in NLV. Aspecten van Informatieverwerking no. 2; ICW-Nota 943